Assignment 3

In [1]:

```
import numpy as np
import matplotlib as mp
import matplotlib.pyplot as plt

from numpy import sin, pi
from scipy.optimize import curve_fit
from textwrap import wrap

%matplotlib inline
%config InlineBackend.figure_format = 'pdf'
```

Question 1: Polynomial fit

We want to find a fourth degree polynomial fit that goes through the points $(-2,0)$, $(-1,-1)$, $(0,0)$, $(1,1)$, and $(2,0)$. To do this, we set up a system of linear equations:

$$\vec{y} = X\vec{a}$$

$$\text{Where} \begin{cases} \vec{y} & \text{is the vector of the "y" values} \\ \vec{X} & \text{is the matrix of the "x" values} \\ \vec{a} & \text{is the vector of the coefficients of the polynomial} \end{cases}$$

Our polynomial will have the form:

$$y = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4$$

Substituting the given values into the equation gives us:

$$\begin{bmatrix} 0 \\ -1 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 & -2 & 4 & -8 & 16 \\ 1 & -1 & 1 & -1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 & 16 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix}$$

To find $\vec{a}$ we first find the inverse of X, then $\vec{a} = X^{-1}\vec{y}$

In [2]:

```
P = np.array([[-2, 0], [-1, -1], [0, 0], [1, 1], [2,0]])
x_space = np.arange(-2, 2, 0.01)
```

In [3]:

```
# Generate matrix "X"
def X(p=P):
    X = np.zeros((p.shape[0], p.shape[0]))
    for i in range(0, p.shape[0]):
```

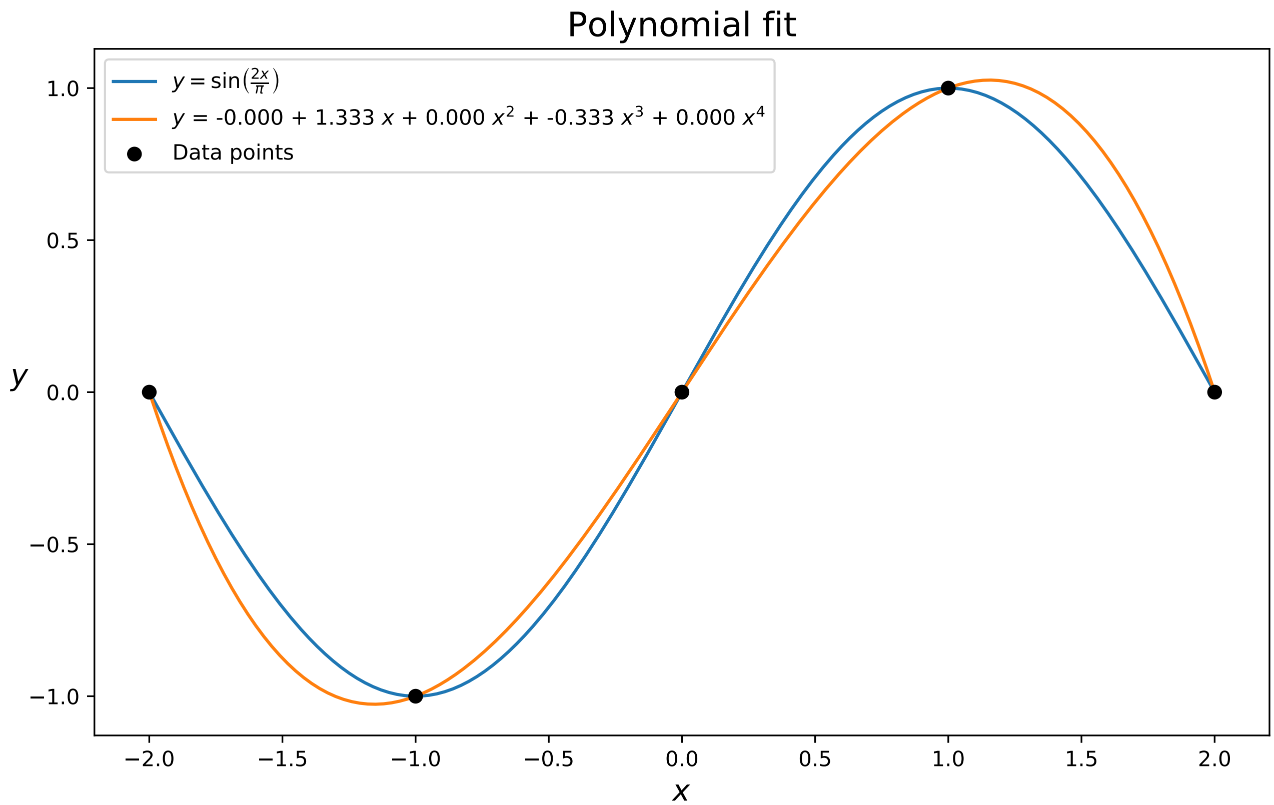
```
        X[i] += p[:,0] ** i
    return X.T

# Invert "X"
def inv_X(p=P):
    inv_x = np.linalg.inv(X(p))
    return inv_x

# Find vector "a"
def vec_a(p=P):
    a =  inv_X(p) @ p[:,1]
    return a
```

In [4]:

```
# Get our vector "a"
a = vec_a()

# Equation for "y"
def y(X, A=a):
    y = 0
    for i in range(0, np.size(A)):
        y += a[i] * X**i
    return y
```

In [5]:

```
# Plot it out:
# ===========

fig, ax = plt.subplots(1, 1, figsize=(10, 6))

ax.scatter(P[:,0], P[:,1], c='black', label="Data points", zorder=5)
ax.plot(x_space, sin(pi*x_space / 2), label=r"$y=\sin \left(\frac{2 x}{\pi}\right)$")
ax.plot(x_space, y(x_space), label=r"$y$ = %5.3f + %5.3f $x$ + %5.3f $x^2$ + %5.3f $x^3$ +
 ↪  %5.3f $x^4$" % tuple(a))
ax.set_ylabel(r"$y$", rotation=0, fontsize=14)
ax.set_xlabel(r"$x$", fontsize=14)
ax.set_title(r"Polynomial fit", fontsize=16)

plt.legend()
plt.show()
```



Question 2: Cubic spline fit

We want to find a cubic spline $S(x)$ with boundary conditions $S'(-2) = S'(2)$ and $S''(-2) = S''(2)$.

As we have five points, we need to find four equations for the functions between these points. This gives us the equation of the cubic spline:

$$\begin{cases} S_0(x_0) = a_0 + b_0x_0 + c_0x_0^2 + d_0x_0^3 \\ S_0(x_1) = a_0 + b_0x_1 + c_0x_1^2 + d_0x_1^3 \\ S_1(x_1) = a_1 + b_1x_1 + c_1x_1^2 + d_1x_1^3 \\ S_1(x_2) = a_1 + b_1x_2 + c_1x_2^2 + d_1x_2^3 \\ S_2(x_2) = a_2 + b_2x_2 + c_2x_2^2 + d_2x_2^3 \\ S_2(x_3) = a_2 + b_2x_3 + c_2x_3^2 + d_2x_3^3 \\ S_3(x_3) = a_3 + b_3x_3 + c_3x_3^2 + d_3x_3^3 \end{cases}$$

We want the spline to have continuous first derivatives:

$$\begin{cases} S_0'(x_1) = b_0 + 2c_0x_1 + 3d_0x_1^2 = b_1 + 2c_1x_1 + 3d_1x_1^2 = S_1'(x_1) \\ S_1'(x_2) = b_1 + 2c_1x_2 + 3d_1x_2^2 = b_2 + 2c_2x_2 + 3d_2x_2^2 = S_2'(x_2) \\ S_2'(x_3) = b_2 + 2c_2x_3 + 3d_2x_3^2 = b_3 + 2c_3x_3 + 3d_3x_3^2 = S_3'(x_3) \end{cases}$$

The boundary conditions require that:

$$S_0'(x_0) = b_0 + 2c_0x_0 + 3d_0x_0^2 = b_3 + 2c_3x_4 + 3d_3x_4^2 = S_3'(x_4)$$

For a natural spline, we want the second order derivatives to be continuous and to be zero at the ends:

$$\begin{cases} S_0''(x_1) = 2c_0 + 6d_0x_1 = 2c_1 + 6d_1x_1 = S_1''(x_1) \\ S_1''(x_2) = 2c_1 + 6d_1x_2 = 2c_2 + 6d_2x_2 = S_2''(x_2) \\ S_2''(x_3) = 2c_2 + 6d_2x_3 = 2c_3 + 6d_3x_3 = S_3''(x_3) \\ S_0''(x_0) = 2c_0 + 6d_0x_0 = 2c_3 + 6d_3x_4 = S_3''(x_4) = 0 \end{cases}$$

Substituting the values into our equations, we have 13 equations to solve for 12 unknowns:

$$\begin{cases} S_0(-2) = 0 = a_0 - 2b_0 + 4c_0 - 8d_0 \\ S_0(-1) = -1 = a_0 - b_0 + c_0 - d_0 \\ S_1(-1) = -1 = a_1 - b_1 + c_1 - d_1 \\ S_1(0) = 0 = a_1 \\ S_2(0) = 0 = a_2 \\ S_2(1) = 1 = a_2 + b_2 + c_2 + d_2 \\ S_3(1) = 1 = a_3 + b_3 + c_3 + d_3 \\ S_3(2) = 0 = a_3 + 2b_3 + 4c_3 + 8d_3 \\ S_0'(-1) = S_1'(-1) = b_0 - 2c_0 + 3d_0 = b_1 - 2c_1 + 3d_1 \\ S_1'(0) = S_2'(0) = b_1 = b_2 \\ S_2'(1) = S_3'(1) = b_2 + 2c_2 + 3d_2 = b_3 + 2c_3 + 3d_3 \\ S_0'(-2) = S_3'(2) = b_0 - 4c_0 + 12d_0 = b_3 + 4c_3 + 12d_3 \\ S_0''(-1) = S_1''(-1) = 2c_0 - 6d_0 = 2c_1 - 6d_1 \\ S_1''(0) = S_2''(0) = 2c_1 = 2c_2 \\ S_2''(1) = S_3''(1) = 2c_2 + 6d_2 = 2c_3 + 6d_3 \\ S_0''(-2) = S_3''(2) = 2c_0 - 12d_0 = 2c_3 + 12d_3 = 0 \end{cases}$$

We can see straight away that $a_1 = a_2 = 0$, $c_0 - 6d_0 = c_3 + 6d_3 = 0$, and $c_2 + 3d_2 = c_3 + 3d_3$, therefore, substitute these into the expressions:

$$\begin{cases} a_0 - 2b_0 + 4c_0 - 8d_0 = 0 \\ a_0 - b_0 + c_0 - d_0 = -1 \\ -b_1 + c_1 - d_1 = -1 \\ a_1 = 0 \\ a_2 = 0 \\ b_2 + c_2 + d_2 = 1 \\ a_3 + b_3 + c_3 + d_3 = 1 \\ a_3 + 2b_3 + 4c_3 + 8d_3 = 0 \\ b_0 - 2c_0 + 3d_0 = b_1 - 2c_1 + 3d_1 \\ b_1 = b_2 \\ b_2 + c_2 = b_3 + c_3 \\ b_0 - 2c_0 = b_3 + 2c_3 \\ c_0 - 3d_0 = c_1 - 3d_1 \\ c_1 = c_2 \\ c_2 + 3d_2 = c_3 + 3d_3 \\ c_0 - 6d_0 = c_3 + 6d_3 = 0 \end{cases}$$

Now, let's put this into a matrix:

$$C\vec{a} = \vec{b}$$

$$\text{Where: } \begin{cases} C \text{ is the matrix containing the coefficient of "a's", "b's", "c's", and "d's"} \\ \vec{a} \text{ is a vector containing the "a's", "b's", "c's", and "d's"} \\ \vec{b} \text{ is a vector containing the number on the right side of the equations} \end{cases}$$

The coefficients can be calculated by:

$$\vec{a} = C^{-1}\vec{b}$$

In [6]:

```
# Create an array "C"
C = np.array([
    [1, -2,  4, -8,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0],
    [1, -1,  1, -1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0],
    [0,  0,  0,  0,  0, -1,  1, -1,  0,  0,  0,  0,  0,  0,  0,  0],
    [0,  0,  0,  0,  0,  0,  0,  0,  0,  1,  1,  1,  0,  0,  0,  0],
    [0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1,  1,  1,  1],
    [0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1,  2,  4,  8],
    [0,  0,  0,  0,  0,  1,  0,  0,  0, -1,  0,  0,  0,  0,  0,  0],
    [0,  0,  0,  0,  0,  0,  1,  0,  0,  0, -1,  0,  0,  0,  0,  0],
    [0,  0,  0,  0,  0,  0,  0,  0,  0,  1,  1,  0,  0, -1, -1,  0],
    [0,  1, -2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, -1, -2,  0],
    [0,  0,  1, -3,  0,  0, -1,  3,  0,  0,  0,  0,  0,  0,  0,  0],
    [0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1,  3,  0,  0, -1, -3],
    [0,  1, -2,  3,  0, -1,  2, -3,  0,  0,  0,  0,  0,  0,  0,  0],
    [0,  0,  1, -6,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, -1, -6],
    [0,  0,  0,  0,  1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0],
    [0,  0,  0,  0,  0,  0,  0,  0,  1,  0,  0,  0,  0,  0,  0,  0]])

# Create an array of vector "b"
vec_b = np.array([
    [0],
    [-1],
    [-1],
    [1],
    [1],
    [0],
    [0],
    [0],
    [0],
    [0],
    [0],
    [0],
    [0],
    [0],
    [0],
    [0]])
```

In [7]:

```
# Find the inverse of "C"
C_inv = np.linalg.inv(C)
```

```
# Take the product of "C" and "b"
vec_a = C_inv @ vec_b
print(r"a = ", vec_a)
```

```
a =  [[ 1.00000000e+00]
 [ 4.50000000e+00]
 [ 3.00000000e+00]
 [ 5.00000000e-01]
 [ 0.00000000e+00]
 [ 1.50000000e+00]
 [ 2.22044605e-16]
 [-5.00000000e-01]
 [ 0.00000000e+00]
 [ 1.50000000e+00]
 [ 2.22044605e-16]
 [-5.00000000e-01]
 [-1.00000000e+00]
 [ 4.50000000e+00]
 [-3.00000000e+00]
 [ 5.00000000e-01]]
```

This is our coefficients for the cubic spline equations.

a) & b): Equations and derivatives of the cubic spline equations

Cubic spline equations:

$$\begin{cases} S_0(x) = 1 + 4.5x + 3x^2 + 0.5x^3 \\ S_1(x) = 1.5x + 2.22 \times 10^{-16}x^2 - 0.5x^3 \\ S_2(x) = 1.5x + 2.22 \times 10^{-16}x^2 - 0.5x^3 \\ S_3(x) = -1 + 4.5x + 3x^2 + 0.5x^3 \end{cases}$$

Derivatives of the cubic spline equations:

$$\begin{cases} S_0(x) = 4.5 + 6x + 1.5x^2 \\ S_1(x) = 1.5 + 4.44 \times 10^{-16}x - 1.5x^2 \\ S_2(x) = 1.5 + 4.44 \times 10^{-16}x - 1.5x^2 \\ S_3(x) = 4.5 + 6x + 1.5x^2 \end{cases}$$

c) Plot it out:

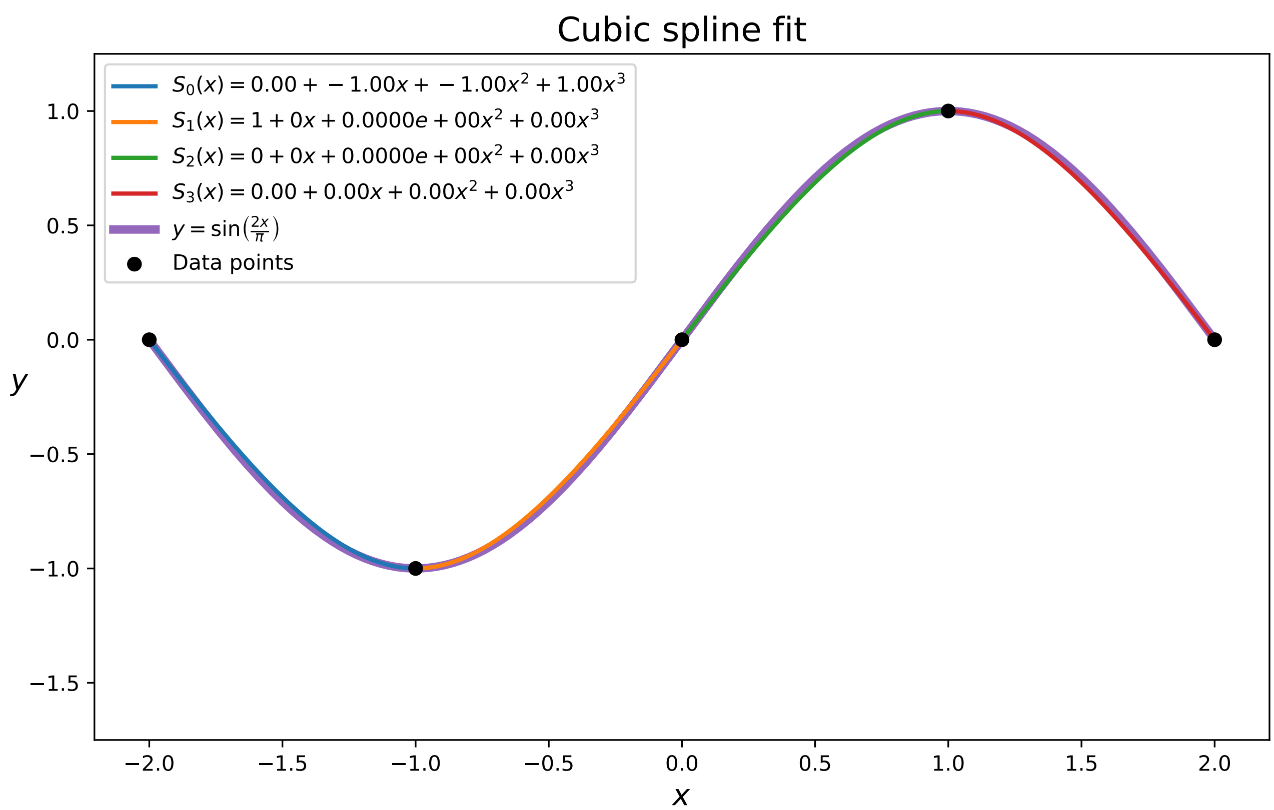
In [8]:

```
x_space0 = np.arange(-2, -1, 0.01)
x_space1 = np.arange(-1, 0, 0.01)
x_space2 = np.arange(0, 1, 0.01)
x_space3 = np.arange(1, 2, 0.01)

def S(A=vec_a, x0=x_space0, x1=x_space1, x2=x_space2, x3=x_space3):
    s0 = A[0] + A[1]*x0 + A[2]*x0**2 + A[3]*x0**3
```

```
    s1 = A[4] + A[5]*x1 + A[6]*x1**2 + A[7]*x1**3
    s2 = A[8] + A[9]*x2 + A[10]*x2**2 + A[11]*x2**3
    s3 = A[12] + A[13]*x3 + A[14]*x3**2 + A[15]*x3**3
    return s0, s1, s2, s3
```

In [9]:

```
# Plot it out:
# ============
S0, S1, S2, S3 = S()

fig, ax2 = plt.subplots(1, 1, figsize=(10, 6))

ax2.scatter(P[:,0], P[:,1], c='black', label="Data points", zorder=5)
ax2.plot(x_space0, S0, label=r"$S_0(x) = {0:.2f} + {1:.2f} x + {2:.2f} x^2 + {3:.2f} x^3$"
         .format(vec_b[0][0], vec_b[1][0], vec_b[2][0], vec_b[3][0]), linewidth=2)
ax2.plot(x_space1, S1, label=r"$S_1(x) = {0} + {1} x + {2:.4e} x^2 + {3:.2f} x^3$"
         .format(vec_b[4][0], vec_b[5][0], vec_b[6][0], vec_b[7][0]), linewidth=2)
ax2.plot(x_space2, S2, label=r"$S_2(x) = {0} + {1} x + {2:.4e} x^2 + {3:.2f} x^3$"
         .format(vec_b[8][0], vec_b[9][0], vec_b[10][0], vec_b[11][0]), linewidth=2)
ax2.plot(x_space3, S3, label=r"$S_3(x) = {0:.2f} + {1:.2f} x + {2:.2f} x^2 + {3:.2f} x^3$"
         .format(vec_b[12][0], vec_b[13][0], vec_b[14][0], vec_b[15][0]), linewidth=2)
ax2.plot(x_space, sin(pi*x_space / 2), label=r"$y=\sin \left(\frac{2 x}{\pi}\right)$",
 ↪  linewidth=4, zorder=0)

ax2.set_ylabel(r"$y$", rotation=0, fontsize=14)
ax2.set_xlabel(r"$x$", fontsize=14)
ax2.set_title(r"Cubic spline fit", fontsize=16)

plt.ylim(-1.75, 1.25)
plt.legend()
plt.show()
```



Question 3

a) Simulate the paths

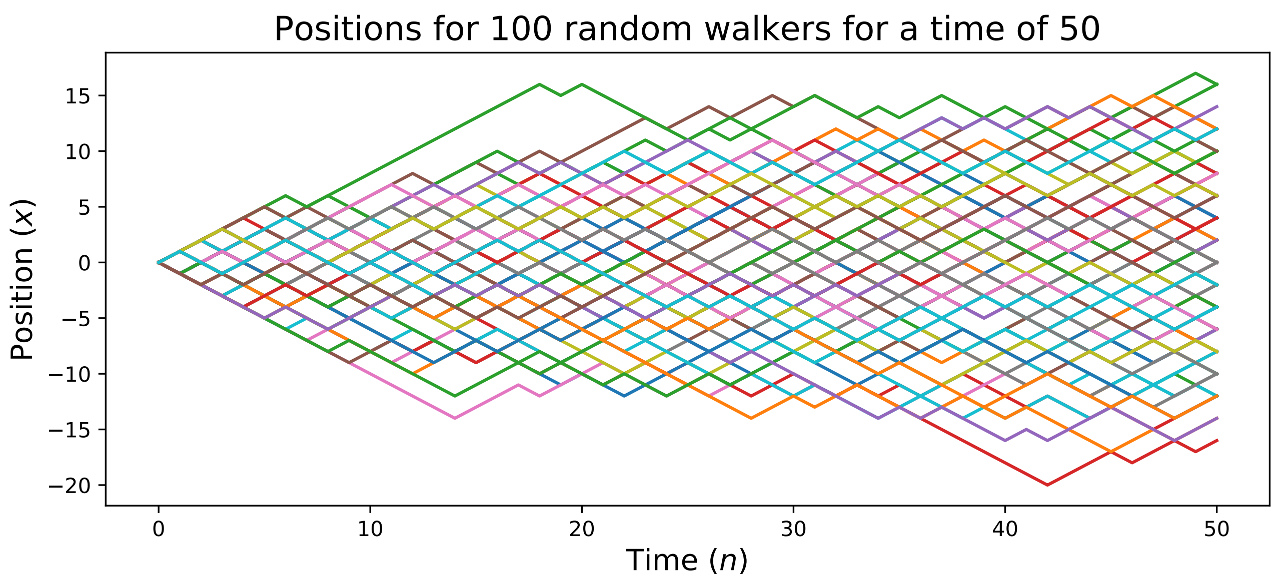
In [10]:

```
paths = 100
time = 50

# Change in steps for random walk
RW = np.random.randint(0, 2, size=(paths, time+1)) * 2 - 1
RW[:, 0] = 0

# Position of the random walkers
RW_pos = np.cumsum(RW, axis=1)

# Time
t = np.arange(0, np.shape(RW_pos)[1], 1)
```

In [11]:

```
fig, ax3 = plt.subplots(1, 1, figsize=(10, 4))

for i in range(0, paths):
    plt.plot(t, RW_pos[i, :])
```

```
    i += 1

plt.title(r"Positions for {0} random walkers for a time of {1}".format(paths, time),
 ↪  fontsize=16)
plt.xlabel(r"Time ($n$)", fontsize=14)
plt.ylabel(r"Position ($x$)", fontsize=14)
plt.show()
```



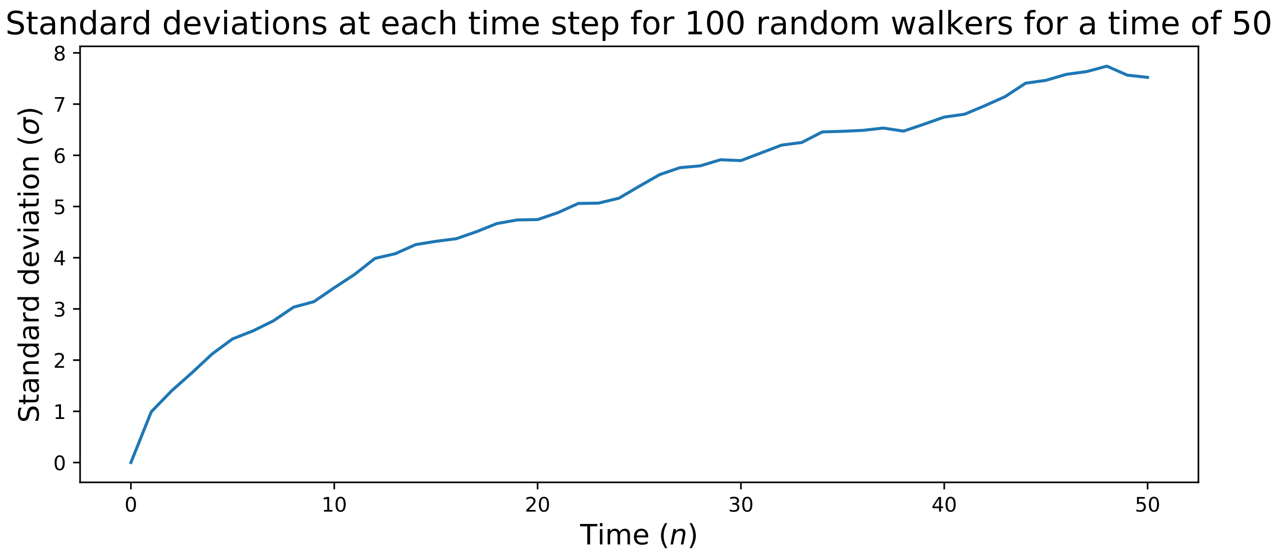
b) Plot the variance for each time step

In [12]:

```
# Find the variance
var = np.nanvar(RW_pos, axis=0)
```

In [13]:

```
fig, ax4 = plt.subplots(1, 1, figsize=(10, 4))

ax4.plot(t, var**0.5)

plt.title(r"Standard deviations at each time step for {0} random walkers for a time of
 ↪  {1}".format(paths, time), fontsize=16)
plt.xlabel(r"Time ($n$)", fontsize=14)
plt.ylabel(r"Standard deviation ($\sigma$)", fontsize=14)
plt.show()
```



c) Show: $\sigma = an^r$

In [14]:

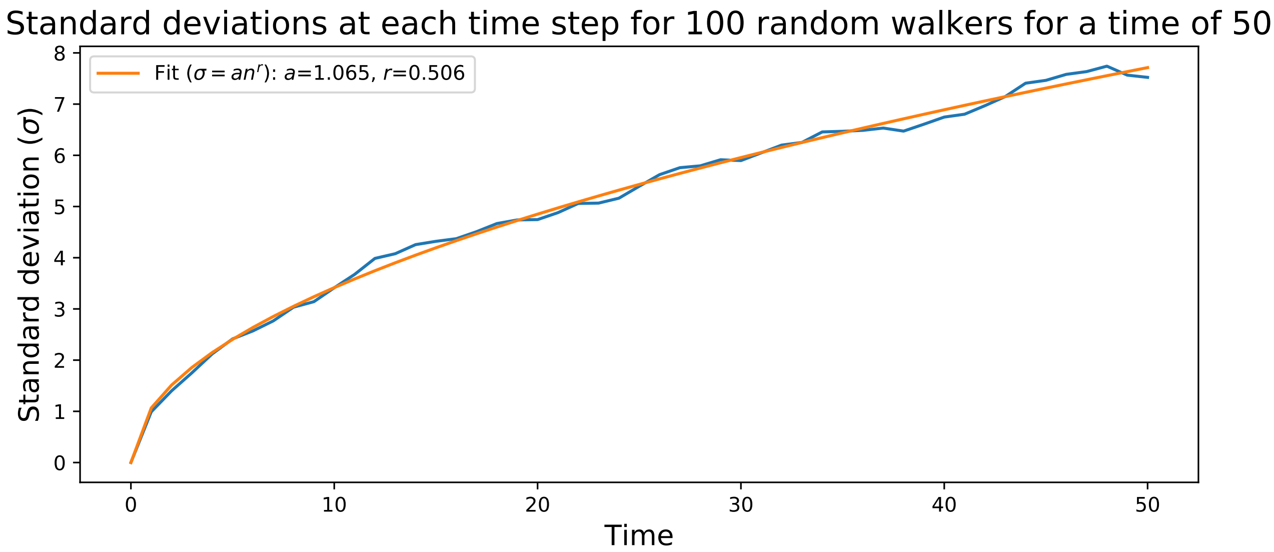
```
# Define function
# ===============
def sdev_fit(t, a, r):
    f = a * t**r
    return f
```

In [15]:

```
# Fit the curve
# =============

popt, pcov = curve_fit(sdev_fit, t, var**0.5)
```

In [16]:

```
fig, ax5 = plt.subplots(1, 1, figsize=(10, 4))

ax5.plot(t, var**0.5)
ax5.plot(t, sdev_fit(t, *popt), '-', label=r'Fit ($\sigma = a n^r$): $a$=%5.3f, $r$=%5.3f' %
 ↪  tuple(popt))

plt.title(r"Standard deviations at each time step for {0} random walkers for a time of
 ↪  {1}".format(paths, time), fontsize=16)
plt.xlabel(r"Time", fontsize=14)
plt.ylabel(r"Standard deviation ($\sigma$)", fontsize=14)
plt.legend()
plt.show()
```



As we can see, the simulation suggests the standard deviation follows:

$$\sigma = an^r$$

$$\text{Where: } \begin{cases} a \approx 0.952 \\ r \approx 0.522 \end{cases}$$